Security Analysis of *SPAKE2+*

Victor Shoup(✉)

New York University, New York, USA
shoup@cs.nyu.edu

Abstract. We show that a slight variant of Protocol *SPAKE2+*, which was presented but not analyzed in [17], is a secure *asymmetric* password-authenticated key exchange protocol (PAKE), meaning that the protocol still provides good security guarantees even if a server is compromised and the password file stored on the server is leaked to an adversary. The analysis is done in the UC framework (i.e., a simulation-based security model), under the computational Diffie-Hellman (CDH) assumption, and modeling certain hash functions as random oracles. The main difference between our variant and the original Protocol *SPAKE2+* is that our variant includes standard key confirmation flows; also, adding these flows allows some slight simplification to the remainder of the protocol. Along the way, we also (i) provide the first proof (under the same assumptions) that a slight variant of Protocol *SPAKE2* from [5] is a secure *symmetric* PAKE in the UC framework (previous security proofs were all in the weaker BPR framework [7]); (ii) provide a proof (under very similar assumptions) that a variant of Protocol *SPAKE2+* that is currently being standardized is also a secure asymmetric PAKE; (iii) repair several problems in earlier UC formulations of secure symmetric and asymmetric PAKE.

1 Introduction

A **password-authenticated key exchange (PAKE)** protocol allows two users who share nothing but a password to securely establish a session key. Ideally, such a protocol prevents an adversary, even one who actively participates in the protocol (as opposed to an eavesdropping adversary), to mount an *offline dictionary attack*. PAKE protocols were proposed initially by Bellovin and Merrit [9], and have been the subject of intensive research ever since.

A formal model of security for PAKE protocols was first proposed by Bellare, Pointcheval, and Rogaway [7]. We call this the **BPR framework for PAKE security**. The BPR framework is a "game based" security definition, as opposed to a "simulation based" security definition. A simulation-based security definition for PAKE was later given in [14]. We shall refer to this and similar simulation-based security definitions as the **UC framework for PAKE security**. Here, UC is short for "Universal Composability", as the definitions in [14] are couched in terms of the more general Universal Composability framework of [12]. As shown in [14], PAKE security in the UC framework implies PAKE

© International Association for Cryptologic Research 2020
R. Pass and K. Pietrzak (Eds.): TCC 2020, LNCS 12552, pp. 31–60, 2020.
https://doi.org/10.1007/978-3-030-64381-2_2

security in the BPR framework. In fact, the UC framework for PAKE security is stronger than the BPR framework in a number of ways that we will discuss further below.

Abdalla and Pointcheval [5] present Protocol *SPAKE2*, which itself is a variant of a protocol originally presented in [9] and analyzed in [7]. Protocol *SPAKE2* is a simple and efficient PAKE protocol, and was shown in [5] to be secure in the BPR security framework. Their proof of security is in the random oracle model [8] under the **computational Diffie-Hellman (CDH) assumption**.[1] The protocol also makes use of a *common reference string* consisting of two random group elements.

Protocol *SPAKE2* has never been proven secure in the UC framework. As we argue below, it seems very unlikely that it can be. One of our results is to show that by adding standard *key confirmation flows* to (a simplified version of) Protocol *SPAKE2* (which is anyway considered to be good security practice), the resulting protocol, which we call Protocol *KC-SPAKE2*, is secure in the UC framework (under the same assumptions).

Protocols *SPAKE2* and *KC-SPAKE2* are *symmetric* PAKE protocols, meaning that both parties must know the password when running the protocol. In the typical setting where one party is a client and the other a server, while the client may memorize their password, the server stores the password in some type of “password file”. If this password file itself is ever leaked, then the client’s password is totally compromised. From a practical security point of view, this vulnerability possibly negates any perceived benefits of using a PAKE protocol instead of a more traditional password-based protocol layered on top of a one-sided authenticated key exchange (which is still the overwhelming practice today).

In order to address this security concern, the notion of an **asymmetric PAKE** was studied in [19], where the UC framework of [14] is extended to capture the notion that after a password file is leaked, an adversary must still carry out an *offline dictionary attack* to retrieve a client’s password. The paper also gives a general mechanism for transforming a secure PAKE into a secure asymmetric PAKE.[2]

In [17], a variant of Protocol *SPAKE2*, called Protocol *SPAKE2+*, is introduced. This protocol is meant to be a secure asymmetric PAKE, while being simpler and more efficient than what would be obtained by directly applying the transformation in [19] to Protocol *SPAKE2* or *KC-SPAKE2*. However, the security of Protocol *SPAKE2+* was never formally analyzed.

In this paper, we propose adding standard key confirmation flows to (a simplified version of) Protocol *SPAKE2+*, obtaining a protocol called Protocol *KC-SPAKE2+*, which we prove is a secure asymmetric PAKE in the UC framework (under the CDH assumption, in the random oracle model, and with

[1] The CDH assumption, in a group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$, asserts that given g^{α}, g^{β}, for random $\alpha, \beta \in \mathbb{Z}_q$, it is hard to compute $g^{\alpha\beta}$.

[2] The paper [19] was certainly not the first to study asymmetric PAKE protocols, nor is it the first to propose a formal security definition for such protocols.

a common reference string). This is our main result. We also present and justify various design choices in both the details of the protocol and the ideal functionality used in its security analysis. As we discuss below, some changes in the ideal functionality in [19] were necessary in order to obtain meaningful results. Since some changes were necessary, we also made other changes in the name of making things simpler.

Comparison to OPAQUE. In [23], a stronger notion of asymmetric PAKE security is introduced, wherein the adversary cannot initiate an offline dictionary attack until *after* the password file is leaked. None of the protocols analyzed here are secure in this stronger sense. Nevertheless, the protocols we analyze here may still be of interest. First, while an offline dictionary attack may be initiated before the password file is leaked, such a dictionary attack must be directed at a particular client. Second, the protocols we analyze here are quite simple and efficient, and unlike the *OPAQUE* protocol in [23], they do not require hashing a password to a group element. Third, the protocols we analyze here are proved secure under the CDH assumption, while the *OPAQUE* protocol is proved secure under the stronger "one-more Diffie-Hellman assumption".

In Defense of Programmable Random Oracles. Our main results are proofs of security in the UC framework using programmable random oracles. The same is true for many other results in this area (including [23]), and results in [20] suggest that secure asymmetric PAKE protocols may only be possible with programmable random oracles.

Recently, results that use programmable random oracles in the UC framework have come to be viewed with some skepticism (see, for example, [11,15]). We wish to argue (briefly) that such skepticism is a bit overblown (perhaps to sell a new "brand" of security) and that such results are still of considerable value.

Besides the fact that in any security analysis the random oracle model is at best a heuristic device (see, for example, [13]), there is a concern that in the UC framework, essential composability properties may be lost. (In fact, this composability concern applies to any type of "programmable" set-up assumption, such as a *common reference string*, and not just to random oracles.)

While composability with random oracles is a concern, in most applications, it is not an insurmountable problem. First, the ideal functionalities we define in this paper will all be explicitly in a multi-user/multi-instance setting where a single random oracle is used for all users and user instances. Second, even if one wants to use the *same* random oracle in this and other protocols, that is not a problem, *so long as all of the protocols involved coordinate on how their inputs are presented to the random oracle*. Specifically, as long as all protocols present their inputs to the random oracle using some convention that partitions the oracle's input space (say, by prefixing some kind of "protocol ID" and/or "protocol instance ID"), there will be no unwanted interactions, and it will be "as if" each different protocol (or protocol instance) is using its own, independent random oracle. In the UC framework, this is all quite easily justified using the

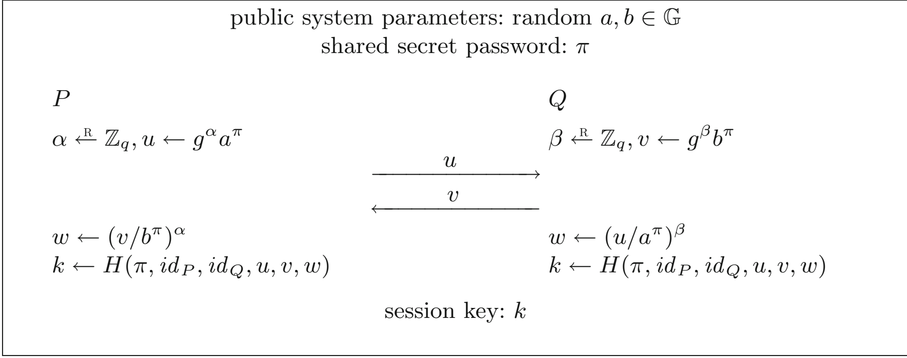

Fig. 1. Protocol *SPAKE2*

JUC theorem [16]. Granted, such coordination among protocols in a protocol stack may be a bit inconvenient, but is not the end of the world.

Full Version of the Paper. Because of space limitations, a number of details have been omitted from this extended abstract. We refer the reader to the full version of the paper [25] for these details.

2 Overview

We start by considering Protocol *SPAKE2*, which is shown in Fig. 1, and which was first presented and analyzed in [5]. Here, $\mathbb{G}$ is a group of prime order q, generated by $g \in \mathbb{G}$, and H is a hash function that outputs elements of the set $\mathcal{K}$ of all possible session keys. Passwords are viewed as elements of $\mathbb{Z}_q$. The protocol also assumes public system parameters $a, b \in \mathbb{G}$, which are assumed to be random elements of $\mathbb{G}$ that are generated securely, so that no party knows their discrete logarithms.

This protocol is perfectly symmetric and can be implemented with the two flows sent in any order or even concurrently. In [5], it was shown to be secure in the BPR framework, under the CDH assumption, and modeling H as a random oracle.

Their security analysis, however, did not take corruption queries (which leak passwords to the adversary) into account, which must be done in order to prove *forward security* in the BPR framework. Later, [1] show that Protocol *SPAKE2* does indeed provide forward security in the BPR framework,[3] also in the random oracle model, but under the stronger *Gap CDH assumption*.[4]

[3] The security theorem in [1] only applies to so-called "weak" corruptions in the BPR framework, in which corrupting a party reveals to the adversary only its password, and not the internal state of any corresponding protocol instance.

[4] The Gap CDH assumption asserts that the problem of computing $g^{\alpha\beta}$, given g^α, g^β for random $\alpha, \beta \in \mathbb{Z}_q$, is hard even if the attacker has access to a *DDH oracle*. Such

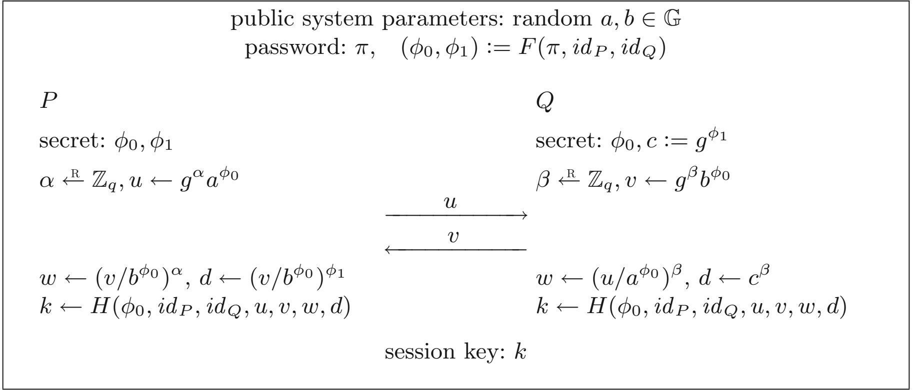

Fig. 2. Protocol *SPAKE2+*

A major drawback of Protocol *SPAKE2* is that if one of the two parties represents a server, and if the server's password file is leaked to the adversary, then the adversary immediately learns the user's password. The initial goal of this research was to analyze the security of Protocol *SPAKE2+*, shown in Fig. 2, which was designed to mitigate against such password file leakage. Protocol *SPAKE2+* was presented in [17], but only some intuition of security was given, rather than a proof. The claim in [17] was that it is secure under the CDH assumption in the random oracle model, but even the security model for this claim was not specified.

The goal is to analyze such a protocol in the model where the password file may be leaked. In the case of such a leakage, the basic security goal is that the adversary cannot log into the server unless it succeeds in an *offline* dictionary attack (note that the adversary can certainly impersonate the server to a client).

In terms of formal models for this setting, probably the best available model is the UC framework for asymmetric PAKE security in [19], which builds on the UC framework for ordinary (i.e., symmetric) PAKE security in [14]. Even without password file leakage, the UC framework is stronger than, and preferable to, the BPR framework in a number of important aspects.

- The UC framework models *arbitrary* password selection, where some passwords may be related, and where the choice of password can be arbitrary, rather than chosen from some assumed distribution. In contrast, the BPR framework assumes that all passwords are independently drawn from some specific distribution.

an oracle is given triples $(g^{\mu}, g^{\nu}, g^{\kappa})$, and returns "yes" if $\kappa = \mu\nu$ and "no" otherwise. This is not a *falsifiable* assumption (as defined in [24]). This is in contrast to the weaker *interactive CDH assumption*, in which it is required that $g^{\mu} = g^{\alpha}$. This is the same assumption used to analyze the well-known DHIES and ECIES schemes (which are essentially just "hashed" ElGamal schemes) in the random oracle model. See [3], where is called the *Strong Diffie-Hellman* assumption.

- In the BPR framework, when the adversary guesses any one password, the game is over and the adversary wins. This means that in a system of 1,000,000 users, if the adversary guesses any one user's password in an online dictionary attack, there are no security guarantees at all for the remaining 999,999 users. In contrast, in the UC framework, guessing one password has no effect on the security of other passwords (to the extent, of course, that those other passwords are independent of the guessed password).
- It is not clear what the security implications of the BPR framework are for secure-channel protocols that are built on top of a secure PAKE protocol. (We will discuss this in more detail in the following paragraphs.) In contrast, PAKE security in the UC framework implies simulation-based security of any secure-channel protocol built on top of the PAKE protocol.

For these reasons, we prefer to get a proof of security in the UC framework. However, Protocol *SPAKE2* itself does not appear to be secure in UC framework for symmetric PAKE (as defined in [14]), and for the same reason, Protocol *SPAKE2+* is not secure in UC framework for asymmetric PAKE (as defined in [19]). One way to see this is as follows. Suppose that in Protocol *SPAKE2* an adversary interacts with Q, and runs the protocol honestly, making a guess that the correct password is π'. Now, at the time the adversary delivers the random group element u to Q, no simulator can have any idea as to the adversary's guess π' (even if it is allowed to see the adversary's queries to the random oracle H), and Q will respond with some v, and from Q's perspective, the key exchange protocol is over, and Q may start using the established session key k in some higher-level secure-channel protocol. For example, at some later time, the adversary might see a message together with a MAC on that message using a key derived from k. At this later time, the adversary can then query H at the appropriate input to determine whether its guess π' was correct or not. However, in the ideal functionality presented in [14], making a guess at a password after the key is established is not allowed. On a practical level, good security practice dictates that any reasonable ideal functionality should not allow this, as any failed online dictionary attack should be detectable by the key exchange protocol. On a more fundamental level, in any reasonable UC formulation, the simulator (or the simulator together with ideal functionality) must decide immediately, at the time a session key is established, whether it is a "fresh" key, a copy of a "fresh" key, or a "compromised" key (one that may be known to the adversary). In the above example, the simulator cannot possibly classify Q's key at the time that the key is established, because it has no way of knowing if the password π' that the adversary has in mind (but which at that time is completely unknown to the simulator) is correct or not. Because it *might* be correct, that would suggest we *must* classify the key as "compromised", even though it may not be. However, if the ideal functionality allowed for that, this would be an unacceptably weak notion of security, as then *every* interaction with the adversary would result in a "compromised" session key.

The obvious way to solve the problem noted above is to enhance Protocol *SPAKE2* with extra *key confirmation flows*. This is anyway considered good

security practice, and the IETF draft specification [26] already envisions such an enhancement.

Note that [1] shows that Protocol *SPAKE2* provides forward security in the BPR framework. This suggests that the notion of forward security defined in [7] is not really very strong; in particular, it does not seem strong enough to prove a meaningful simulation-based notion of security for a channel built on top of a PAKE protocol.

In concurrent and independent work, [2] show that Protocol *SPAKE2* is secure in the UC framework, but with respect to a weak ideal functionality (which, again, does not seem strong enough to prove a meaningful simulation-based notion of security for a channel built on top of a PAKE protocol). They also show that Protocol *SPAKE2* with additional key confirmation flows is secure in the UC framework, with respect to an ideal functionality very similar to that considered here. We note that all of their security proofs make use of the very strong *Gap* CDH assumption, mentioned above, whereas all of our results only make use of the *standard* CDH assumption.

2.1 Outline

In Sect. 3, we introduce Protocol *KC-SPAKE2*, which is a variation of Protocol *SPAKE2* that includes key confirmation. In Sect. 4 we give a fairly self-contained overview of the general UC framework, and in Sect. 4.2, we specify the symmetric PAKE ideal functionality that we will use to analyze Protocol *KC-SPAKE2* in the UC framework (discussing why we made certain changes to the UC formulation in [14]). In Sect. 5, we introduce Protocol *KC-SPAKE2+*, which is to Protocol *KC-SPAKE2* as Protocol *SPAKE2+* is to Protocol *SPAKE2*. In Sect. 6, we specify the *asymmetric* PAKE ideal functionality that we will use to analyze Protocol *KC-SPAKE2+* in the UC framework, and we discuss why we made certain changes to the UC formulation in [19]. Section 7 describes Protocol *IETF-SPAKE2+*, which is a variant of Protocol *KC-SPAKE2+* that generalizes the protocol described in the IETF draft specification [26]. Section 8 gives formal statements of our security theorems. All of our security proofs are in the random oracle model under the CDH assumption (with some additional, standard assumptions on some symmetric primitives for Protocol *IETF-SPAKE2+*). Although we do not have space in this extended abstract to present proofs of these theorems (but which are presented in the full version of the paper [25]), in Sect. 9 we present a very brief sketch of some of the main ideas.

3 Protocol *KC-SPAKE2*

We begin by presenting a protocol, *KC-SPAKE2*, which is a simplified version of Protocol *SPAKE2* with key confirmation flows. This protocol is essentially Protocol *PFS-SPAKE2* presented in the paper [6]. In that paper, this protocol was shown to provide perfect forward secrecy in the BPR framework [7] (under the

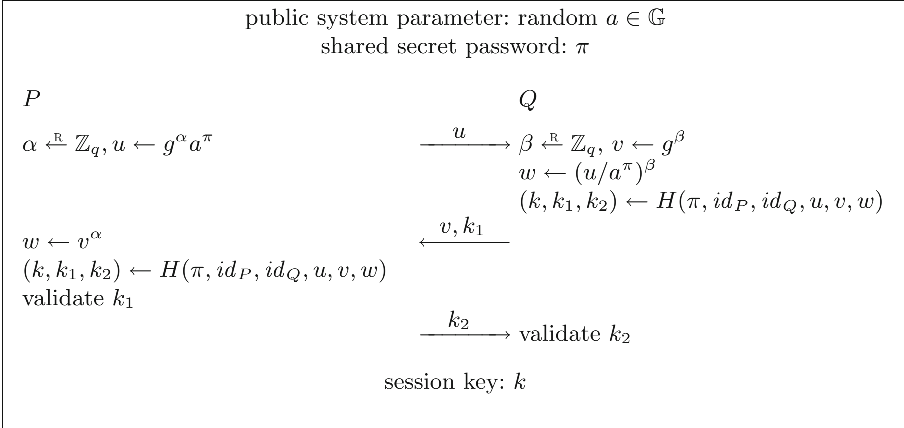

Fig. 3. Protocol *KC-SPAKE2*

CDH assumption in the random oracle model). Our goal here is to analyze Protocol *KC-SPAKE2* in the UC model, and then to augment Protocol *KC-SPAKE2* so that it is secure against server compromise in the UC framework. Note that in the paper [4], a protocol that is very similar to Protocol *KC-SPAKE2* was also shown to provide perfect forward secrecy in the BPR framework (also under the CDH assumption in the random oracle model).

Protocol *KC-SPAKE2* makes use of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in G$. It also makes use of a hash function H, which we model as a random oracle, and which outputs elements of the set $\mathcal{K} \times \mathcal{K}_{\text{auth}} \times \mathcal{K}_{\text{auth}}$, where $\mathcal{K}$ is the set of all possible session keys, and $\mathcal{K}_{\text{auth}}$ is an arbitrary set of super-polynomial size, used for explicit key confirmation. The protocol has a public system parameter $a \in \mathbb{G}$, which is assumed to be a random element of $\mathbb{G}$ that is generated securely, so that no party knows its discrete logarithm. Furthermore, passwords are viewed as elements of $\mathbb{Z}_q$. Protocol *KC-SPAKE2* is described in Fig. 3. Both users compute the value $w = g^{\alpha\beta}$, and then compute $(k, k_1, k_2) \leftarrow H(\pi, id_P, id_Q, u, v, w)$. Note that P "blinds" the value g^{α} by multiplying it by a^{π}, while Q does not perform a corresponding blinding. Also, P checks that the authentication key k_1' it receives is equal to the authentication key k_1 that it computed; otherwise, P aborts *without sending* k_2. Likewise, Q checks that the authentication key k_2' it receives is equal to the authentication key k_2 that it computed; otherwise, Q aborts. The value k is the session key.

In this protocol, it is essential that the P first sends the flow u, and then Q responds with v, k_1, and only then (if k_1 is valid) does P respond with k_2. Also, in this protocol, as well as Protocol *KC-SPAKE2+* (in Sect. 5), it is essential that P validates that $v \in \mathbb{G}$ and Q validates that $u \in \mathbb{G}$.

It is useful to think of P as the client and Q as the server. From a practical point of view, this is a very natural way to assign roles: the client presumably initiates any session with the server, and the first flow of Protocol *KC-SPAKE2*

can piggyback on that initial message. In addition, as we transition from Protocol *KC-SPAKE2* to Protocol *KC-SPAKE2+*, we will also assign P the role of client and Q the role of server.

Detecting Failed Online Dictionary Attacks. From a practical perspective, it is desirable to be able to detect a failed online dictionary attack and to take preventive action. In Protocol *KC-SPAKE2*, a client P can detect a (potential) failed online dictionary attack when it receives an invalid authentication key k_1 (of course, the authentication key could be invalid for other, possibly benign, reasons, such as a transmission error). Moreover, the adversary can only learn if its password guess was correct by seeing how P responds. Indeed, the adversary learns nothing at all if it does not send the second flow to P. Now consider how a server Q may detect a (potential) failed online dictionary attack. After the server sends out k_1, the adversary can already check if its guess was correct. If its guess was incorrect, it cannot feasibly respond with a valid k_2, and such an adversary would presumably not even bother sending k_2. Thus, if Q times out waiting for k_2, then to be on the safe side, Q must consider such a time-out to be a potential online dictionary attack. If, from a security perspective, it is viewed that online dictionary attacks against the server are more likely, it might be advantageous to flip the roles of P and Q, so that it is the client that sends the first authentication key. Unfortunately, in the typical setting where the client sends the first flow, this will increase the number of flows from 3 to 4. Although we do not analyze this variant or its asymmetric secure "+" variant, it should be straightforward to modify the proofs presented here to cover these variants as well. Finally, as we already mentioned above, in the original Protocol *SPAKE2*, which provides no explicit key confirmation, it is impossible to detect a failed online dictionary attack in the key exchange protocol itself.

4 Simulation-Based Definition of Secure PAKE

Protocol *KC-SPAKE2* was already analyzed in the BPR framework. Our goal is to analyze Protocol *KC-SPAKE2* in the UC framework. The main motivation for doing so is that we eventually want to analyze the asymmetric Protocol *KC-SPAKE2+* in the UC framework.

We give a fairly self-contained definition of a secure PAKE. Our definition is a simulation-based definition that is essentially in the UC framework of [12]. We do not strictly adhere to all of the low-level mechanics and conventions given in [12]. Indeed, it is not really possible to do so, for a couple of reasons. First, between the time of its original appearance on the eprint archive and the time of this writing, the paper [12] has been revised a total of 14 times, with some of those revisions being quite substantial. So it is not clear what "the" definition of the UC framework is. Second, as pointed out in [22] and [21], the definitions in the contemporaneous versions of [12] were mathematically inconsistent. While there are more recent versions of [12], we have not yet been able to independently validate that these newer versions actually correct the problems identified in [22]

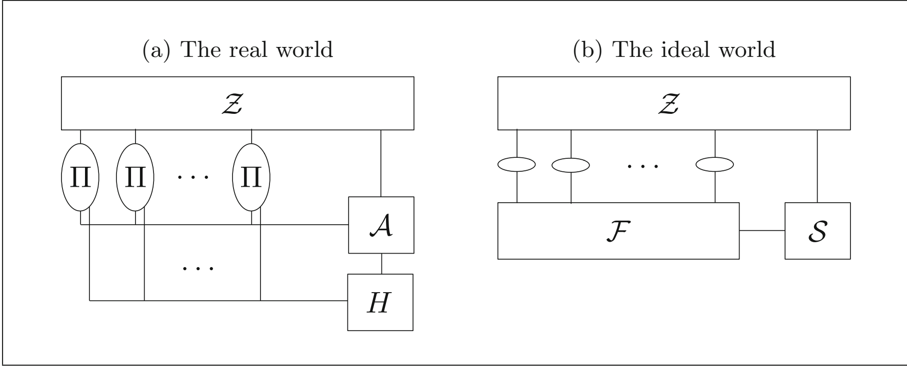

Fig. 4. The real and ideal worlds in the UC framework

and [21], while not introducing new problems. Our point of view, however, is that even though it is extremely difficult to get all of the details right, the core of the UC framework is robust enough so as to give meaningful security guarantees even if some of the low-level mechanics are vaguely or even inconsistently specified, and that these security guarantees are mainly independent of small changes to these low-level mechanics. In fact, it is fair to say that most papers that purport to prove results in the UC framework are written without any serious regard toward, or even knowledge of, most of these low-level mechanics.

Our definitions of security for PAKE differ from that presented in [14], which was the first paper to formally define secure PAKE in the UC framework. Some of these differences are due to the fact that we eventually will modify this functionality to deal with server corruptions, and so we will already modify the definition to be more compatible with this. While the paper [19] builds on [14] to model asymmetric PAKE, the paper [20] identifies several flaws in the model of [19]. Thus, we have taken it upon ourselves to correct these flaws in a reasonable way.

4.1 Review of the UC Framework

We begin with a very brief, high-level review of the UC framework. Figure 4a shows a picture of the "real world" execution of a Protocol Π. The oval shapes represent individual machines that are faithfully executing the protocol Π. The environment $\mathcal{Z}$ represents higher-level protocols that use Π as a sub-protocol, as well as any adversary that is attacking those higher-level protocols. However, all of these details are abstracted away, and $\mathcal{Z}$ can be quite arbitrary. The adversary $\mathcal{A}$ represents an adversary attacking Protocol Π. Adversary $\mathcal{A}$ communicates continuously with $\mathcal{Z}$, so as to coordinate its attack on Π with any ongoing attack on a higher-level protocol. The protocol machines receive their inputs from $\mathcal{Z}$ and send their outputs to $\mathcal{Z}$. Normally, one would think of these inputs as coming from and going to a higher-level protocol. The protocol machines also send and receive messages from $\mathcal{A}$, but not with each other. Indeed, among other things,

the adversary $\mathcal{A}$ essentially models a completely insecure network, effectively dropping, injecting, and modifying protocol messages at will. Figure 4a also shows a box labeled H. In our analysis of Protocol *KC-SPAKE2*, we model H a random oracle. This means that in the "real world", the protocol machines and the adversary $\mathcal{A}$ may directly query the random oracle H. The environment $\mathcal{Z}$ does not have direct access to H; however, it can access H indirectly via $\mathcal{A}$.

Figure 4b shows a picture of the "ideal world" execution. The environment $\mathcal{Z}$ is exactly the same as before. The box labeled $\mathcal{F}$ is an *ideal functionality* that is essentially a trusted third party that we *wish* we could use to run the protocol for us. The small oval shapes also represent protocol machines, but now these protocol machines are just simple "repeaters" that pass their inputs directly from $\mathcal{Z}$ to $\mathcal{F}$, and their outputs directly from $\mathcal{F}$ to $\mathcal{Z}$. The box labeled $\mathcal{S}$ is called a *simulator*, but really it is just an adversary that happens to operate in the "ideal world". The simulator $\mathcal{S}$ can converse with $\mathcal{Z}$, just as $\mathcal{A}$ did in the "real world". The simulator $\mathcal{S}$ can also interact with $\mathcal{F}$, but its influence on $\mathcal{F}$ will typically be limited: the precise influence that $\mathcal{S}$ can exert on $\mathcal{F}$ is determined by the specification of $\mathcal{F}$ itself. Typically, while $\mathcal{S}$ cannot cause any "bad events" that would violate security, it can still determine the order in which various events occur.

Roughly speaking, we say that Protocol Π *securely emulates* ideal functionality $\mathcal{F}$ if for every efficient adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$, such that no efficient environment $\mathcal{Z}$ can effectively distinguish between the "real world" execution and the "ideal world" execution. The precise meaning of "efficient" here is a variant of polynomial time that adequately deals with a complex, multi-party system. We suggest the definitions in [21,22], but other definitions are possible as well. In the UC framework, saying that Protocol Π is "secure" means that it securely emulates $\mathcal{F}$. Of course, what "secure" means depends on the specification of $\mathcal{F}$.

4.2 An Ideal Functionality for PAKE

We now give our ideal functionality for PAKE. As mentioned above, the functionality we present here is a bit different from that in [14], and some of the low-level mechanics (relating to things like "session identifiers") is a bit different from those in [12].

- Party P inputs: $(\texttt{init-client}, \mathit{rid}, \pi)$
  Intuition: This models the initialization of a client and its relationship to a particular server, including the shared password π.
    - We say that P is *initialized as a client*, where *rid* is its *relationship ID* and π is its *password*.
    - Assumes (i) that P has not been previously initialized as either a client or server, and (ii) that no other client has been initialized with the same relation ID.[5]

[5] As we describe it, the ideal functionality imposes various pre-conditions on the inputs it receives. The reader may assume that if these are not met, an "error message" back to whoever sent the input. However, see Remark 1 below.

- ◦ The simulator is sent $(\texttt{init-client}, P, rid)$.
  ◦ Note that rid is a relationship ID that corresponds to a single client/server pair. In practice (and in the protocols analyzed here), such a relationship ID is a pair $rid = (id_P, id_Q)$.

- Party Q inputs: $(\texttt{init-server}, rid, \pi)$
  Intuition: This models the initialization of a server and its relationship to a particular client, including the shared password π.
  ◦ We say that Q is *initialized as a server*, where rid is its *relationship ID* and π is its *password.*
  ◦ Assumes (i) that Q has not been previously initialized at either a client or server, and (ii) that no other server has been initialized with the same relationship ID.
  ◦ Assumes that if a client and server are both initialized with the same relationship ID rid, then they are both initialized with the same password π.
  ◦ The simulator is sent $(\texttt{init-server}, Q, rid)$.
  ◦ Note: for any relationship ID, there can be at most one client and one server with that ID, and we call this client and server *partners.*

- Party P inputs: $(\texttt{init-client-instance}, iid_P, \pi^*)$
  Intuition: This models the initialization of a client instance, which corresponds to a single execution of the key exchange protocol by the client, using possibly mistyped or misremembered password π^*.
  ◦ Party P must have been previously initialized as a client.
  ◦ The value iid_P is an *instance ID*, and must be unique among all instances of P.
  ◦ The simulator is sent $(\texttt{init-client-instance}, P, iid_P, type)$, where $type := 1$ if $\pi^* = \pi$, and otherwise $type := 0$, and where π is P's password.
  ◦ We call this instance (P, iid_P), and the ideal functionality sets the state of the instance to *original.*
  ◦ We call π^* the *password of this instance*, and we say that this instance is *good* if $\pi^* = \pi$, and *bad* otherwise.
  ◦ Note: a *bad* client instance is meant to model the situation in the actual, physical world where the human client mistypes or misremembers their password associated with the server.

- Party Q inputs: $(\texttt{init-server-instance}, iid_Q)$
  Intuition: This models the initialization of a server instance, which corresponds to a single execution of the key exchange protocol by the server.
  ◦ Party Q must have been previously initialized as a server.
  ◦ The value iid_Q is an *instance ID*, and must be unique among all instances of Q.
  ◦ The simulator is sent $(\texttt{init-server-instance}, Q, iid_Q)$.
  ◦ We call this instance (Q, iid_Q), and the ideal functionality sets the state of the instance to *original.*

- ◦ If π is Q's password, we also define $\pi^* := \pi$ to be the *password of this instance*. Unlike client instances, server instances are always considered *good*.

- Simulator inputs: $(\texttt{test-pwd}, X, iid_X, \pi')$
Intuition: This models an *on-line* dictionary attack, whereby an attacker makes a *single* guess at a password by interacting with a particular client/server instance.
  - ◦ Assumes (i) that there is an instance (X, iid_X), where X is either a client or server, (ii) that this is the first test-pwd for (X, iid_X), and (iii) that the state of (X, iid_X) is either *original* or *abort*.
  - ◦ The ideal functionality tests if π' is equal to the password π^* of instance (X, iid_X):
    - – if $\pi' = \pi^*$, then the ideal functionality does the following: (i) if the state of the instance is *original*, it changes the state to *correct-guess*, and (ii) sends the message (correct) to the simulator.
    - – if $\pi' \neq \pi^*$, then the ideal functionality does the following: (i) if the state of the instance is *original*, it changes the state to *incorrect-guess*, and (ii) sends the message (incorrect) to the simulator.
  - ◦ Note: if X is a server or (X, iid_X) is a *good* client instance, then $\pi^* = \pi$, where π is X's password.

- Simulator inputs: $(\texttt{fresh-key}, X, iid_X, sid)$
Intuition: This models the successful termination of a protocol instance that returns to the corresponding client or server a *fresh* key, i.e., a key that is completely random and independent of all other keys and of the attacker's view, along with the given *session ID sid*. This is *not* allowed if a password guess was made against this instance.
  - ◦ Note: if X is a server or (X, iid_X) is a *good* client instance, then $\pi^* = \pi$, where π is X's password. The value sid is a *session ID* that is to be assigned to the instance (X, iid_X).
  - ◦ Assumes (i) that (X, iid_X) is an *original, good* instance, where X is either a client or a server, (ii) that there is no other instance (X, iid'_X) that has been assigned the same session ID sid, (iii) that X has a partner Y, and (iv) that there is no instance (Y, iid_Y) that has been assigned the same session ID sid.
  - ◦ The ideal functionality does the following: (i) assigns the session ID sid to the instance (X, iid_X), (ii) generates a random session key k, (iii) changes the state of the instance (X, iid_X) to *fresh-key*, and (iv) sends the output $(\texttt{key}, iid_X, sid, k)$ to X.

- Simulator inputs: $(\texttt{copy-key}, X, iid_X, sid)$
Intuition: This models the successful termination of a protocol instance that returns to the corresponding client or server a *copy* of a fresh key, along with the given *session ID sid*. Note that a fresh key can be copied only once and only from an appropriate partner instance with a matching session ID. This is *not* allowed if a password guess was made against this instance.

- Assumes (i) that (X, iid_X) is an *original, good* instance, where X is either a client or server, (ii) that there is no other instance (X, iid'_X) that has been assigned the same session ID *sid*, (iii) that X has a partner Y, (iv) that there is a unique instance (Y, iid_Y) that has been assigned the same session ID *sid*, and (v) the state of (Y, iid_Y) is *fresh-key*.
- The ideal functionality does the following: (i) assigns the session ID *sid* to the instance (X, iid_X), (ii) changes the state of the instance (X, iid_X) to *copy-key*, and (iii) sends the output $(\texttt{key}, iid_X, sid, k)$ to X, where k is the key that was previously generated for the instance (Y, iid_Y).

• Simulator inputs: $(\texttt{corrupt-key}, X, iid_X, sid, k)$
Intuition: This models the successful termination of a protocol instance that returns to the corresponding client or server a *corrupt* key, i.e., a key that is known to the adversary, along with the given *session ID sid*. This is *only* allowed if a corresponding password guess against this *particular* instance was successful.

- Assumes (i) that (X, iid_X) is a *correct-guess* instance, where X is either a client or server, (ii) that there is no other instance (X, iid'_X) that has been assigned the same session ID *sid*, and (iii) that if X has a partner Y, there is no instance (Y, iid_Y) that has been assigned the same session ID *sid*.
- The ideal functionality does the following: (i) assigns the session ID *sid* to the instance (X, iid_X), (ii) changes the state of the instance (X, iid_X) to *corrupt-key*, and (iii) sends the output $(\texttt{key}, iid_X, sid, k)$ to X.

• Simulator inputs: $(\texttt{abort}, X, iid_X)$
Intuition: This models the unsuccessful termination of a protocol instance. Note that an incorrect password guess against an instance can only lead to its unsuccessful termination.

- Assumes that (X, iid_X) is either an *original*, *correct-guess*, or *incorrect-guess* instance, where X is either a client or server.
- The ideal functionality does the following: (i) changes the state of the instance (X, iid_X) to *abort*, and (ii) sends the output $(\texttt{abort}, iid_X)$ to X.

4.3 Well-Behaved Environments

In the above specification of our ideal functionality, certain pre-conditions must be met on inputs received from the environment (via the parties representing clients and servers). To this end, we impose certain restrictions on the environment itself.

We say that an environment $\mathcal{Z}$ is **well behaved** if the inputs from clients and servers (which come from $\mathcal{Z}$) do not violate any of the stated preconditions. Specifically, this means that for the `init-client` and `init-server` inputs: (i) no two clients are initialized with same relationship ID, (ii) no two servers are initialized with the same relationship ID, and (iii) if a client and server are initialized with the same relationship ID, then they are initialized with the same password; moreover, for the `init-client-instance` and `init-server` inputs

(iv) no two instances of a given client or server are initialized with the same instance ID.

In formulating the notion that a concrete protocol "securely emulates" the ideal functionality, one restricts the quantification over all environments to all such *well-behaved* environments. It is easy to verify that all of the standard UC theorems, including dummy-adversary completeness, transitivity, and composition, hold when restricted to *well-behaved* environments.

Remark 1. In describing our ideal functionality, in processing an input from a client, server, or simulator, we impose pre-conditions on that input. In all cases, these pre-conditions can be efficiently verified by the ideal functionality, and one may assume that if these pre-conditions are not satisfied, then the ideal functionality sends an "error message" back to whoever sent it the input.

However, it is worth making two observations. First, for inputs from a client or server, these pre-conditions cannot be "locally" validated by the given client or server; however, the assumption that the environment is well-behaved guarantees that the corresponding pre-conditions will always be satisfied (see Remark 2 below for further discussion). Second, for inputs from simulator, the simulator itself has enough information to validate these pre-conditions, and so without loss of generality, we can also assume that the simulator does not bother submitting invalid inputs to the ideal functionality.

4.4 Liveness

In general, UC security by itself does not ensure any notion of protocol "liveness". For a PAKE protocol, it is natural to define such a notion of liveness as follows. In the real world, if the adversary faithfully delivers all messages between a *good* instance I of a client P and an instance J of P's partner server Q, then I and J will both output a session key and their session IDs will match. All of the protocols we examine here satisfy this notion of liveness.

With our PAKE ideal functionality, UC security implies that if an instance I of a client P and an instance J of P's partner server Q both output a session key, and their session IDs match, then one of them will hold a "fresh" session key, while the other will hold a copy of that "fresh" key.

If we also assume liveness, then UC security implies the following. Suppose that the adversary faithfully delivers all messages between a *good* instance I of a client P and an instance J of P's partner server Q. Then I and J will both output a session key, their session IDs will match, and one of them will hold a "fresh" session key, while the other will hold a copy of that "fresh" key. Moreover, by the logic of our ideal functionality, this implies that in the ideal world, the simulator did not make a guess at the password. See further discussion in Remark 7 below.

4.5 Further Discussion

Remark 2. As in [14], our ideal functionality does not specify how passwords are chosen or how a given clients/server pair come to agree upon a shared password.

All of these details are relegated to the environment. Our "matching password restriction", which says that in any well-behaved environment (Sect. 4.3), a client and server that share the same relationship ID must be initialized with the same password, really means this: *whatever the mechanism used for a client and server to agree upon a shared password, the agreed-upon password should be known to the client (resp., server) before the client (resp., server) actually runs an instance of the protocol.*

This "matching password restriction" seems perfectly reasonable and making it greatly simplifies both the logic of the ideal functionality and the simulators in our proofs.

Note that the fact that client inputs this shared password to the ideal functionality during client initialization is not meant to imply that in a real protocol the client actually stores this password anywhere. Indeed, in the actual, physical world, we expect that a human client may memorize their password and not store it anywhere (except during the execution of an instance of the protocol). Our model definitely allows for this.

Also note that whatever mechanisms are used to choose a password and share it between a client and server, as well as to choose relationship IDs and instance IDs, these mechanisms must satisfy the requirements of a well-behaved environment. These requirements are quite reasonable and easy to satisfy with well-known techniques under reasonable assumptions.

Remark 3. Our formalism allows us to model the situation in the actual, physical world where a human client mistypes or misremembers their password. This is the point of having the client pass in the password π^* when it initializes a client instance. The "matching password restriction" (see Remark 2) makes it easy for the ideal functionality to immediately classify a client as *good* or *bad*, according to whether or not $\pi^* = \pi$, where π is the password actually shared with the corresponding server.[6] The logic of our ideal functionality implies that the only thing that can happen to a *bad* instance is that either: (a) the instance aborts, or (b) the adversary makes one guess at π^*, and if that guess is correct, the adversary makes that instance accept a "compromised" key. In particular, no instance of the corresponding server will ever share a session ID or session key with a *bad* client instance.

Mistyped or misremembered passwords are also modeled in [14] and subsequent works (such as [19] and [20]). All of these works insist on "hiding" from the adversary, to some degree or other, whether or not the client instance is *bad*. It is not clear what the motivation for this really is. Indeed, in [14], they observe that "we are not aware of any application where this is needed". Moreover, in the typical situation where a client is running a secure-channels protocol on top of a PAKE protocol, an adversary will almost inevitably find out that a client

[6] Otherwise, if the corresponding server had not yet been initialized with a password π at the time this client instance had been initialized with a password π^*, the ideal functionality could not determine (or inform the simulator) whether or not $\pi^* = \pi$ at that time. This would lead to rather esoteric complications in the logic of the ideal functionality and the simulators in our proofs.

instance is *bad*, because it will most likely abort without a session key (or possibly, as required in [14], will end up with a session key that is not shared with any server instance).

So to keep things simple, and since there seems little motivation to do otherwise, our ideal simply notifies the simulator if a client instance is *good* or *bad*, and it does so immediately when the client instance is initialized. Indeed, as pointed out in [20], the mechanism [19] for dealing with mistyped or misremembered passwords was flawed. In [20], another mechanism is proposed, but our mechanism is much simpler and more direct.

Remark 4. One might ask: why is it necessary to explicitly model mistyped or misremembered passwords at all? Why not simply absorb *bad* client instances into the adversary. Indeed, from the point of view of preventing such a *bad* client from logging into a server, this is sufficient. However, it would not adequately model security for the client: if, say, a human client enters a password that is nearly identical to the correct password, this should not compromise the client's password in any way; however, we cannot afford to model this situation by giving this nearly-identical password to the adversary.

Note that the BPR framework [7] does not model mistyped or misremembered passwords at all. We are not aware of any protocols that are secure in the BPR framework that become blatantly insecure if a client enters a closely related but incorrect password.

Remark 5. In our formalism, in the real world, all instances of a given client are executed by a single client machine. This is an abstraction, and should not be taken too literally. In the real, physical world, a human client may choose to run instances of the protocol on different devices. Logically, there is nothing preventing us from mapping those different devices onto the same client machine in our formalism.

Remark 6. In our formalism, in the real world, a server instance must be initialized (by the environment) before a protocol message can be delivered (from the adversary) to that instance. This is an abstraction, and should not be taken too literally. In practice, a client could initiate a run of the protocol by sending an initial message over the network to the server, who would then initialize an instance of the protocol and then effectively let the adversary deliver the initial message to that instance.

Remark 7. As in all UC formulations of PAKE, the simulator (i.e., ideal-world adversary) gets to make at most one password guess per protocol instance, which is the best possible, since in the real world, an adversary may always try to log in with a password guess. Moreover, as discussed above in Sect. 4.4, then assuming the protocol provides liveness, the simulator does not get to make any password guesses for protocol executions in which the adversary only eavesdrops. This corresponds to the "Execute" queries in the BPR framework, in which an adversary passively eavesdrops on protocol executions, and which do not increase the odds of guessing a password. Unlike the formulation in [14], where this property requires a proof, this property is explicitly built in to the definition.

Remark 8. Our ideal functionality is explicitly a "multi-session" functionality: it models all of the parties in the system and all runs of the protocol.

Formally, for every client/server pair (P, Q) that share a relationship ID *rid*, this ID will typically be of he form $rid = (id_P, id_Q)$, where id_P is a client-ID and id_Q is a server-ID. This is how relationship IDs will be presented Protocol *KC-SPAKE2*, but it is not essential. In practice, the same client-ID may be associated with one user in relation to one server, and with a different user in relation to another server.

For a given party X, which may either be a client P or server Q, it will have associated with it several *instances*, each of which has an *instance ID* iid_X. Note that in the formal model, identifiers like P and Q denote some kind of formal identifier, although these are never intended to be used in any real protocols. Similarly, instance IDs are also not intended to be used in any real protocols. These are all just "indices" used in the formalism to identify various participants. It is the relationship IDs and session IDs that are meant to be used by and have meaning in higher-level protocols. Looking ahead, the session IDs for *KC-SPAKE2* will be the partial conversations (u, v).

Also note that every instance of a server Q in our formal model establishes sessions with instances of the same client P. In practice, of course, a "server" establishes sessions with many clients. One maps this onto our model by modeling such a "server" as a collection of several of our servers.

Remark 9. What we call a *relationship ID* corresponds to what is called a "session ID" in the classical UC framework [12]. Our ideal functionality explicitly models many "UC sessions"—this is necessary, as we eventually need to consider several such "UC sessions" since all of the protocols we analyze make use of common reference string and random oracles shared across many such "UC sessions". What we call a *session ID* actually corresponds most closely what is called a "subsession ID" in [14] (in the "multi-session extension" of their PAKE functionality). Note that [14], a client and server instance have to agree in advance on a "subsession ID". This is actually quite impractical, as it forces an extra round of communication just to establish such a "subsession ID". In contrast, our *session IDs* are computed as part of the protocol itself (which more closely aligns with the notion of a "session ID" in the BPR framework [7]).

In our model, after a session key is established, a higher-level protocol would likely use a string composed of the *relationship ID*, the *session ID*, and perhaps other elements, as a "session ID" in the sense of [12].

Remark 10. Our ideal functionality models explicit authentication in a fairly strict sense. Note that [14] does not model explicit authentication at all. Furthermore, as pointed out in [20], the formulation of explicit authentication in [19] is flawed. Our ideal functionality is quite natural in that when an adversary makes an unsuccessful password guess on a protocol instance, then when that instance terminates, the corresponding party will receive an `abort` message. Our formulation of explicit authentication is similar to that in [20], but is simpler because (as discussed above in Remark 3) we do not try to hide the fact that a

client instance is *bad*. Another difference is that in our formulation, the simulator may first force an abort and then only later make its one password guess—this behavior does not appear to be allowed in the ideal functionality in [20]. This difference is essential to be able to analyze *KC-SPAKE2*, since an adversary may start a session with a server, running the protocol with a guessed password, but after the server sends the message v, k_1, the adversary can send the server some garbage, forcing an abort, and then at a later time, the adversary may evaluate the random oracle at the relevant point to see if its password guess was correct.

Remark 11. We do not explicitly model corrupt parties, or corruptions of any kind for that matter (although this will change somewhat when we model server compromise in the asymmetric PAKE model in Sect. 6). In particular, all client and server instances in the real world are assumed to faithfully follow their prescribed protocols. This may seem surprising, but it is not a real restriction. First of all, anything a statically corrupt party could do could be done directly by the adversary, as there are no authenticated channels in our real world. In addition, because the environment manages passwords, our formulation models adaptively exposing passwords, which corresponds to the "weak corruption model" of the BPR framework [7]. Moreover, just like the security model in [14], our security model implies security in the "weak corruption model" of the BPR framework.[7] The proof is essentially the same as that in [14]. However, just like in [14] (as well as in [19] and [20]), our framework does not model adaptive corruptions in which an adversary may obtain the internal state of a running protocol instance.[8]

5 Protocol *KC-SPAKE2+*

We present Protocol *KC-SPAKE2+* in Fig. 5. Given a password π, a client derives a pair $(\phi_0, \phi_1) \in \mathbb{Z}_q^2$ using a hash function F, which we model as a random oracle. The server, on the other hand, just stores the pair (ϕ_0, c), where $c := g^{\phi_1} \in \mathbb{G}$. Note that unlike Protocol *KC-SPAKE2*, in Protocol *KC-SPAKE2+*, a password π need not be an element of $\mathbb{Z}_q$, as it first gets passed through the hash function F.

6 An Ideal Functionality for Asymmetric PAKE

First, as we already noted, the attempt to formulate an asymmetric PAKE functionality in [19] was fundamentally flawed, as was demonstrated in [20]. One major problem identified in [20] was that after a server is compromised, we need

[7] Actually, our framework does not model the notion in [7] that allows password information stored on the server to be changed. That said, we are ultimately interested asymmetric PAKE, and we are not aware of any asymmetric PAKE functionality in the literature that models this notion.

[8] This type of corruption would correspond to the "strong corruption model" of the BPR framework [7]. Note that the protocol analyzed in [7] is itself only proven secure in the "weak corruption model".

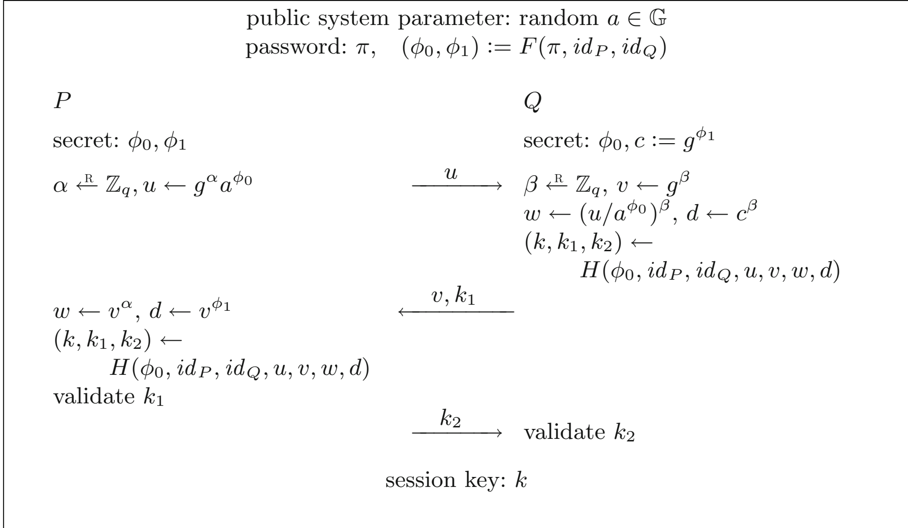

Fig. 5. Protocol *KC-SPAKE2+*

a good way to bound the number of "offline test" queries in the ideal world in terms of the number of "random oracle" queries in the real world. The paper [20] points out that the ideal functionality suggested in [19] cannot actually be realized by *any* protocol (including the protocol presented in [19]) for which the "password file record" stored on the server is efficiently and deterministically computable from the password. However, the "fix" proposed in [20] relies in an essential way on the notion of polynomial running time in the UC framework as formulated in [12], and as pointed out in [22], this notion of running time is itself flawed (and may or may not have been repaired in later versions of [12]). Moreover, ignoring these technical problems, the "fix" in [20] is not very satisfactory, as it does not yield a *strict* bound on the number of "offline test" queries in terms of the number "random oracle" queries. Rather, it only guarantees that the simulator runs in time bounded by some polynomial in the number of bits passed to the adversary from the environment.

We propose a simple and direct way of dealing with this issue. It is a somewhat protocol-specific solution, but it gets the job done, and is hopefully of more general utility. We assume that the protocol in question makes use of a hash function F, which we model as a random oracle, and that inputs to F are of the form (π, rid), where rid is a relationship ID, and π is a password. However, the ideal functionality for accessing this random oracle is a bit non-standard. Specifically, in the real world, the *adversary* is not allowed direct access to this random oracle. Rather, for the adversary to obtain the output value of the random oracle at some input, the *environment* must specifically instruct the random oracle functionality to give this output value to the adversary. More precisely, the environment may send an input message $(\texttt{oracle-query}, \pi, rid)$ to the random oracle functionality,

who responds by sending the message ($\texttt{oracle-query}, \pi, rid, F(\pi, rid)$) to the adversary.[9] Note that machines representing clients and servers may access F directly.

Now, in our ideal functionality for asymmetric PAKE, when the environment sends an input ($\texttt{oracle-query}, \pi, rid$), this is sent to the asymmetric PAKE functionality, who simply forwards the message ($\texttt{oracle-query}, \pi, rid$) to the simulator.[10] *Moreover*, when the simulator makes an "offline test" query to the ideal functionality, the ideal functionality will only allow such a query if there was already a corresponding `oracle-query`. This simple mechanism restricts the number of "offline line" test queries made against a particular rid in the ideal world strictly in terms of the number of "random oracle" queries made with the same rid in the real world.

In addition to supporting the `oracle-query` interface discussed above, the following changes are made to the ideal functionality in Sect. 4.2. There is a new interface:

- Server Q inputs: ($\texttt{compromise-server}$)
  - The simulator is sent ($\texttt{compromise-server}, Q$).
  - We say that Q is *compromised.*

Note that in the real world, upon receiving ($\texttt{compromise-server}$), server Q sends to the adversary its "password file record" for this particular client/server pair (for Protocol *KC-SPAKE2+*, this would be the pair (ϕ_0, c)).[11] However, the server Q otherwise continues to faithfully execute its protocol as normal. There is a second new interface, which allows for "offline test" queries:

- Simulator inputs: ($\texttt{offline-test-pwd}, Q, \pi'$)
  - Assumes (i) that Q is a compromised server, and (ii) that the environment has already submitted the query ($\texttt{oracle-query}, rid, \pi'$) to the ideal functionality, where rid is Q's relationship ID.
  - The simulator is sent ($\texttt{correct}$) if $\pi' = \pi$, and ($\texttt{incorrect}$) if $\pi' \neq \pi$.

Finally, to model the fact that once a server Q is corrupted, the simulator is always able to impersonate the server to its partner P, we modify the `corrupt-key` interface as follows. Specifically, condition (i), which states:

(i) that (X, iid_X) is a *correct-guess* instance, where X is either a client or server

is replaced by the following:

(i) that either: (a) (X, iid_X) is a *correct-guess* instance, where X is either a client or server, or (b) (X, iid_X) is an *original, good* instance and X's partner is a compromised server

7 Protocol *IETF-SPAKE2+*

[9] Note that in the specific UC framework of [12], the environment sends this message to the random oracle functionality via a special "dummy" party.

[10] This allows the simulator to "program" the random oracle.

[11] As in [21], we model this type of compromise simply by a message sent from the environment, rather than the more indirect mechanism in [12].

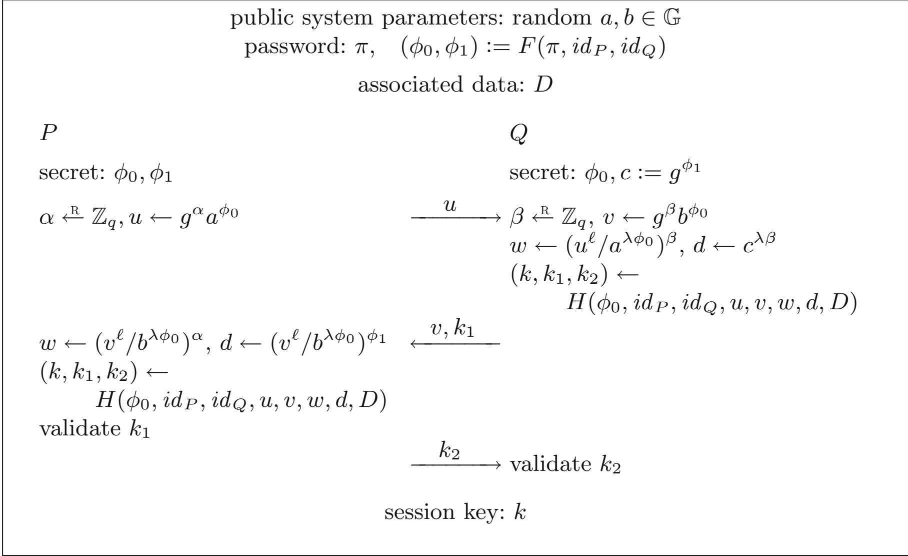

Fig. 6. Protocol *IETF-SPAKE2+*

Here we describe Protocol *IETF-SPAKE2+*, which is generalization of the protocol called *SPAKE2+* in the IETF draft specification [26]. The protocol is presented in Fig. 6. Unlike the previous PAKE protocols we have presented, both client and server take as input **associated data** D, and the session key is computed as (u, v, D). We will discuss below in detail the semantic significance of this associated data.

Unlike in Protocol *KC-SPAKE2+*, group elements received by a party may lie in some larger group $\overline{\mathbb{G}}$ containing $\mathbb{G}$ as a subgroup. It is assumed that parties validate membership in $\overline{\mathbb{G}}$ (which is generally cheaper than validating membership in $\mathbb{G}$). Specifically, P should validate that $v \in \overline{\mathbb{G}}$ and Q should validate that $u \in \overline{\mathbb{G}}$. We also assume that the index of $\mathbb{G}$ in $\overline{\mathbb{G}}$ is ℓ, where ℓ is not divisible by q. Note that for all $x \in \overline{\mathbb{G}}$, we have $x^{\ell} \in \mathbb{G}$.

We shall denote by λ the image of ℓ in $\mathbb{Z}_q$. Note that $\lambda \neq 0$. We will be rather careful in our notation involving exponents on group elements. Namely, on elements in $\mathbb{G}$, exponents will always be elements of $\mathbb{Z}_q$. On elements that lie in $\overline{\mathbb{G}}$ but not necessarily in $\mathbb{G}$, the only exponent that will be used is the index ℓ.

Unlike in Protocol *KC-SPAKE2+*, the functions F and H in Protocol *IETF-SPAKE2+* are not modeled as random oracles; rather, they are defined as follows:

$$\begin{aligned} &F(\pi, id_P, id_Q) := (\phi_0, \phi_1) \\ &\qquad \text{where} \quad (\phi_0, \phi_1) := F_1(f) \quad \text{and} \quad f := F_0(\pi, id_P, id_Q). \end{aligned} \tag{1}$$

and

$$H(\phi_0, id_P, id_Q, u, v, w, d, D) := (k, k_1, k_2),$$
$$\text{where} \quad k_1 := H_1(h; 1, u, D), \quad k_2 := H_1(h; 2, v, D) \qquad (2)$$
$$\text{and} \quad (k, h) := H_0(\phi_0, id_P, id_Q, u, v, w, d).$$

Here, the functions F_0, F_1, H_0, H_1 are modeled as follows:

- F_0 is modeled as a random oracle producing an output $f \in \mathcal{F}$, where $\mathcal{F}$ is some large finite set.
- F_1 is modeled as a pseudorandom generator (PRG) with seed space $\mathcal{F}$ and output space $\mathbb{Z}_q \times \mathbb{Z}_q$.
- H_0 is modeled as a random oracle producing an output $(k, h) \in \mathcal{K} \times \mathcal{H}$, where $\mathcal{H}$ is some large finite set.
- H_1 is modeled as a pseudorandom function (PRF) with key space $\mathcal{H}$, input space $\{1, 2\} \times \overline{\mathbb{G}} \times \mathcal{D}$, and output space $\mathcal{K}_{\text{auth}}$. As before, we assume that the size of $\mathcal{K}_{\text{auth}}$ is super-polynomial.

Remark 12. We can actually prove security under significantly weaker assumptions on F_1 and H_1. See the full version of the paper [25] for details.

Remark 13. The second public parameter $b \in \mathbb{G}$ is not necessary, and all of our proofs of security hold even if the discrete log of b is fixed and/or known to the adversary. In particular, one can set $b = 1$.

Remark 14. In the IETF draft specification [26], the function F_0 is defined to be *PBKFD*, which outputs two bit strings p_0 and p_1, and $F_1(p_0, p_1) = (\phi_0, \phi_1) = (p_0 \bmod q, p_1 \bmod q)$, where p_0 and p_1 are viewed as the binary representations of integers. Here, *PBKFD* is a *password-based key derivation function* designed to slow down brute-force attackers. Examples are *Scrypt* [RFC7914] and *Argon2* [10]. The inputs to *PBKFD* are encoded using a prefix-free encoding scheme. The lengths of p_0 and p_1 should be significantly longer than the bit length of q to ensure that ϕ_0 and ϕ_1 have distributions that are statistically close to uniform on $\mathbb{Z}_q$. In order to minimize the reliance on random oracles, it is also possible to incorporate the final stage of *PBKFD* in the function F_1.

Remark 15. In the IETF draft specification, the function H_0 is defined to be a *hash function Hash*, which may be *SHA256* or *SHA512* [RFC6234]. The inputs to *Hash* are encoded using a prefix-free encoding scheme. The output of *Hash* is a bit string $h \parallel k$. The computation of the function $H_1(h; i, x, D)$, where $i \in \{1, 2\}$, $x \in \overline{\mathbb{G}}$, and $D \in \mathcal{D}$, is defined as follows. First, we compute

$$(h_2 \parallel h_1) = \mathit{KDF}(\mathit{nil}, h, \texttt{"ConfirmationKeys"} \parallel D).$$

Here, *KDF* is a key derivation function such as *HKDF* [RFC5869], which takes as input a salt (here, *nil*), intermediate keying material (here, h), info string (here, `"ConfirmationKeys"` $\parallel D$), as well as a derived-key-length parameter (not shown here). The output of H_1 is $\mathit{MAC}(h_i, x)$, where *MAC* is a *message*

authentication code, such as *HMAC* [RFC2104] or *CMAC-AES-128* [RFC4493], which takes as input a key (here, h_i) and a message (here, x). In the modes of operation used here, it is reasonable to view *KDF* as a PRF (with key h), and to view *MAC* (with key h_i) as a PRF. Assuming both of these are PRFs implies that H_1 itself is a PRF.

Remark 16. We model the above implementations of H_0 and F_0 as independent random oracles. Ideally, this would be verified by carrying out a complete analysis in the indifferentiability framework [18]. As proved in [18], the implementation of H_0 as a standard hash function, like *SHA256* or *SHA512*, with prefix-free input encoding, is indeed a random oracle in the indifferentiability framework, under appropriate assumptions on the underlying compression function. We are not aware of an analogous analysis for the implementation of F_0 as a standard password-based key derivation function, like *Scrypt*, or of any possible interactions between the hash functions used in both (which may be the same). Also, for H_0, it would be preferable that its inputs were prefixed with some sort of protocol ID, and that higher-level protocols that use the same hash function similarly prefix their inputs with an appropriate protocol ID. (which includes the system parameters a and b). This would ensure that there are no unwanted interactions between random oracles used in different protocols. Similarly, it would be preferable if F_0 were implemented by first hashing its input using the same hash function used for H_0, but prefixed with a different protocol ID (and which includes the system parameters a and b). This would ensure no unwanted interactions between these two random oracles. Despite these recommendations, it seems highly unlikely that the current IETF draft specification [26] has any real weaknesses.

Remark 17. The IETF draft specification allows id_P and/or id_Q to be omitted as inputs to H_0 and F_0 under certain circumstances. Our analysis does not cover this.

Remark 18. The IETF draft specification allows the parties to negotiate a ciphersuite. Our analysis does not cover this. We assume a fixed ciphersuite is used by all parties throughout the lifetime of the protocol.

Remark 19. Including u and v as inputs to H_1 is superfluous, and could have been omitted without any loss in security. Equivalently, in the IETF draft specification discussed above in Remark 15, we could just set $(k_1, k_2) := (h_1, h_2)$, and forgo the *MAC* entirely.

Remark 20. The IETF draft specification insists that the client checks that $v^\ell \neq 1$ and that the server checks that $u^\ell \neq 1$. This is superfluous. We will ignore this. Indeed, one can easily show that a protocol that makes these checks securely emulates one that does not. Note, however, that by making these checks, the "liveness property" (see Sect. 4.4) only holds with overwhelming probability (rather than with probability 1).

Remark 21. The IETF draft specification allows the server to send v and k_1 as separate messages. We will ignore this. Indeed, one can easily show that a protocol that sends these as separate messages securely emulates one that does not. This is based on the fact that in the IETF draft specification, even if v and k_1 are sent by the server as separate messages, they are sent by the server only after it receives u, and the client does nothing until it receives both v and k_1.

7.1 Extending the Ideal Functionality to Handle Associated Data

The only change required to deal with associated data are the `init-client-instance` and `init-client-instance` interfaces. In both cases, the associated data D is passed along as an additional input to the ideal functionality, which in turn passes it along immediately as an additional output to the simulator.

Remark 22. In practice, the value of D is determined outside of the protocol by some unspecified mechanism. The fact that D is passed as an input to `init-client-instance` and `init-server-instance` means that D must be fixed for that instance before the protocol starts. The fact that the value is sent to the simulator means that the protocol does not treat D as private data.

Remark 23. Our ideal functionality for PAKE allows a client instance and a server instance to share a session key only if their session IDs are equal. For Protocol *IETF-SPAKE2+*, since the session ID is computed as (u, v, D), the ideal functionality will allow a client instance and a server instance to share a session key only if their associated data values are equal. In fact, as will be evident from our proof of security, in Protocol *IETF-SPAKE2+*, if an adversary faithfully delivers messages between a client instance and a server instance, but their associated data values do not match, then neither the client nor the server instance will accept any session key at all. Our ideal functionality does not model this stronger security property.

8 Statement of Main Results

Our main results are the following:

Theorem 1. *Under the CDH assumption for $\mathbb{G}$, and modeling H as a random oracle, Protocol KC-SPAKE2 securely emulates the ideal functionality in Sect. 4.2 (with respect to all well-behaved environments as in Sect. 4.3).*

Theorem 2. *Under the CDH assumption for $\mathbb{G}$, and modeling H and F as random oracles, Protocol KC-SPAKE2+ securely emulates the ideal functionality in Sect. 6 (with respect to all well-behaved environments as in Sect. 4.3).*

Theorem 3. *Under the CDH assumption for $\mathbb{G}$, assuming F_1 is a PRG, assuming H_1 is a PRF, and modeling H_0 and F_0 as random oracles, Protocol IETF-SPAKE2+ securely emulates the ideal functionality in Sect. 6 (with respect to all well-behaved environments as in Sect. 4.3, and with associated data modeled as in Sect. 7.1).*

Because of space limitations, we refer the reader to the full version of the paper [25] for proofs of these theorems. In the full version of the paper, we also briefly discuss alternative proofs under an the interactive CDH assumption, which yield tighter reductions.

9 Sketch of Proof Ideas

Although we do not have space to provide detailed proofs, we can give a sketch of some of the main ideas.

Protocol *KC-SPAKE2*. We start by giving an informal argument that Protocol *KC-SPAKE2* is a secure symmetric PAKE under the CDH assumption, and modeling H as a random oracle. We make use of a "Diffie-Hellman operator", defined as follows: for $\alpha, \beta \in \mathbb{Z}_q$, define

$$[g^\alpha, g^\beta] = g^{\alpha\beta}. \tag{3}$$

We first make some simple observations about this operator. For all $x, y, z \in \mathbb{G}$ and all $\mu, \nu \in \mathbb{Z}_q$, we have

$$[x, y] = [y, x], \quad [xy, z] = [x, z][y, z], \quad \text{and} \quad [x^\mu, y^\nu] = [x, y]^{\mu\nu}.$$

Also, note that $[x, g^\mu] = x^\mu$, so given any two group elements x and y, if we know the discrete logarithm of either one, we can efficiently compute $[x, y]$.

Using this notation, the CDH assumption can be stated as follows: given random $s, t \in \mathbb{G}$, it is hard to compute $[s, t]$.

First, consider a passive adversary that eavesdrops on a run of the protocol between an instance of P and an instance of Q. He obtains a conversation (u, v, k_1, k_2). The session key and authentication values are computed by P and Q is

$$(k, k_1, k_2) = H(\pi, id_P, id_Q, u, v, [u/a^\pi, v]). \tag{4}$$

Intuitively, to mount an offline dictionary attack, the adversary's goal is to query the random oracle H at as many *relevant* points as possible, where here, a relevant point is one of the form

$$(\pi', id_P, id_Q, u, v, [u/a^{\pi'}, v]), \tag{5}$$

where $\pi' \in \mathbb{Z}_q$. By evaluating H at relevant points, and comparing the outputs to the values k_1, k_2 (as well as values derived from k), the adversary can tell whether or not $\pi' = \pi$.

The following lemma shows that under the CDH assumption, he is unable to make even a single relevant query:

Lemma 1. *Under the CDH assumption, the following problem is hard: given random $a, u, v \in \mathbb{G}$, compute $\gamma \in \mathbb{Z}_q$ and $w \in \mathbb{G}$ such that $w = [u/a^\gamma, v]$.*

Proof. Suppose we have an adversary that can efficiently solve the problem in the statement of the lemma with non-negligible probability. We show how to use this adversary to solve the CDH problem with non-negligible probability. Given a challenge instance (s, t) for the CDH problem, we compute

$$\mu \xleftarrow{R} \mathbb{Z}_q, \quad a \leftarrow g^\mu,$$

and then we give the adversary

$$a, \quad u := s, \quad v := t.$$

Suppose now that the adversary computes for us $\gamma \in \mathbb{Z}_q$ and $w \in \mathbb{G}$ such that $w = [u/a^\gamma, v]$. Then we have

$$w = [u, v][a, v]^{-\gamma} \tag{6}$$

Since we know the discrete log of a, we can compute $[a, v]$, and therefore, we can compute $[u, v] = [s, t]$

□

Next, consider an active adversary that engages in the protocol with an instance of server Q.

Now, in the adversary's attack, he submits the first message u to Q. Next, Q chooses v at random and sends this to the adversary. Server Q also computes k, k_1, k_2 as in (4) and also sends k_1 to the adversary. Again, the adversary's goal is to evaluate the random oracle H at as many relevant points, as in (5), as possible. Of course, an adversary that simply follows the protocol using some guess π' for the password can always make one relevant query. What we want to show is that it is infeasible to make more than one relevant query. This is implied by the following lemma:

Lemma 2. *Under the CDH assumption, the following problem is hard: given random $a, v \in \mathbb{G}$, compute $\gamma_1, \gamma_2 \in \mathbb{Z}_q$ and $u, w_1, w_2 \in \mathbb{Z}_q$ such that $\gamma_1 \neq \gamma_2$ and $w_i = [u/a^{\gamma_i}, v]$ for $i = 1, 2$.*

Proof. Suppose that we are given an instance (s, t) of the CDH problem. We give the adversary

$$a := s, \quad v := t.$$

The adversary computes for us γ_1, γ_2 and w_1, w_2 such that $\gamma_1 \neq \gamma_2$, and

$$w_i = [u/a^{\gamma_i}, v] = [u, v][a, v]^{-\gamma_i} \quad (i = 1, 2).$$

Then we have

$$w_2/w_1 = [a, v]^{\gamma_1 - \gamma_2}. \tag{7}$$

This allows us to compute $[a, v] = [s, t]$. □

Note that if an adversary tries to mount a dictionary attack by interacting with an instance of a client P, by design, the adversary gets only one guess at the password: the only random oracle query that matters is the one that yields the value k_1 that the adversary sends to the client instance.

Protocol *KC-SPAKE2+*. It is not hard to argue that Protocol *KC-SPAKE2+* offers the same level of security as protocol *KC-SPAKE2* under normal conditions, when the server is not compromised. However, consider what happens if the server Q is compromised in protocol *KC-SPAKE2+*, and the adversary obtains ϕ_0 and c. At this point, the adversary could attempt an offline dictionary attack, as follows: evaluate F at points (π', id_P, id_Q) for various passwords π', trying to find π' such that $F(\pi', id_P, id_Q) = (\phi_0, \cdot)$. If this succeeds, then with high probability, $\pi' = \pi$, and the adversary can easily impersonate the client P.

The key property we want to prove is the following: if the above dictionary attack fails, then under the CDH assumption, the adversary cannot impersonate the client.

To prove this property, first suppose that an adversary compromises the server, then attempts a dictionary attack, and finally, attempts to log in to the server. Compromising the server means that the adversary obtains ϕ_0 and $c = g^{\phi_1}$. Now suppose the dictionary attack fails, which means that the adversary has not evaluated F at the point (π, id_P, id_Q). The value ϕ_1 is completely random, and the adversary has no other information about ϕ_1, other than the fact that $c = g^{\phi_1}$. When he attempts to log in, he sends the server Q some group element u', and the server responds with $v := g^{\beta}$ for random $\beta \in \mathbb{Z}_q$. To successfully impersonate the client, he must explicitly query the random oracle H at the point $(\phi_0, id_P, id_P, u', v, [u'/a^{\phi_0}, v], [c, v])$, which means, in particular, he has to compute $[c, v]$. But since, from the adversary's point of view, c and v are random group elements, computing $[c, v]$ is tantamount to solving the CDH problem.

The complication we have not addressed in this argument is that the adversary may also interact with the client P at some point, giving some arbitrary message (v', k_1') to an instance of P, and in the above algorithm for solving the CDH, we have to figure out how the CDH solver should respond to this message. Assuming that (v', k_1') did not come from an instance of Q, the only way that P will not abort is if k_1' is the output of a query to H explicitly made by the adversary; moreover, this query must have been $(\phi_0, id_P, id_P, u, v', [u/a^{\phi_0}, v'], [c, v'])$, where u is the random group element generated by this instance of P. Now, with overwhelming probability, there is at most one query to H that outputs k_1'; however, we have to determine if it is of the required form. We may assume that our CDH solver knows $\log_g a$ (in addition to ϕ_0), and so our CDH solver needs to be able to determine, given adversarially chosen $v', w', d' \in \mathbb{G}$, whether or not $w' = [u, v']$ and $d' = [c, v']$. Since it does not know $\log_g c$, our CDH solver would appear to need an oracle to answer such queries. Without the additional condition $w' = [u, v']$, we would require the interactive CDH assumption; however, with this additional condition, we can use the "Twin Diffie-Hellman Trapdoor

Test" from [17] to efficiently implement this oracle, and so we only need the *standard* CDH assumption.

References

1. Abdalla, M., Barbosa, M.: Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194 (2019). https://eprint.iacr.org/2019/1194
2. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. Cryptology ePrint Archive, Report 2020/320 (2020). https://eprint.iacr.org/2020/320
3. Abdalla, M., Bellare, M., Rogaway, P.: The Oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45353-9_12
4. Abdalla, M., Bresson, E., Chevassut, O., Möller, B., Pointcheval, D.: Provably secure password-based authentication in TLS. In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, pp. 35–45 (2006). https://doi.org/10.1145/1128817.1128827
5. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. CT-RSA **2005**, 191–208 (2005)
6. Becerra, J., Ostrev, D., Skrobot, M.: Forward secrecy of spake2. Cryptology ePrint Archive, Report 2019/351 (2019). https://eprint.iacr.org/2019/351
7. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. Cryptology ePrint Archive, Report 2000/014 (2000). https://eprint.iacr.org/2000/014
8. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS 1993, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, 3–5 November 1993, pp. 62–73. ACM (1993). https://doi.org/10.1145/168588.168596
9. Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, 4–6 May 1992, pp. 72–84 (1992)
10. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: the memory-hard function for password hashing and other applications (2017). https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf
11. Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 280–312. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_11
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000). https://eprint.iacr.org/2000/067
13. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM **51**(4), 557–594 (2004)
14. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. Cryptology ePrint Archive, Report 2005/196 (2005). https://eprint.iacr.org/2005/196

15. Canetti, R., Jain, A., Scafuro, A.: Practical UC security with a global random oracle. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014, pp. 597–608 (2014)
16. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_16
17. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. Cryptology ePrint Archive, Report 2008/067 (2008). https://eprint.iacr.org/2008/067
18. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: how to construct a hash function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_26
19. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_9
20. Hesse, J.: Separating standard and asymmetric password-authenticated key exchange. Cryptology ePrint Archive, Report 2019/1064 (2019). https://eprint.iacr.org/2019/1064
21. Hofheinz, D., Shoup, V.: GNUC: a new universal composability framework. J. Cryptol. **28**(3), 423–508 (2015). https://doi.org/10.1007/s00145-013-9160-y
22. Hofheinz, D., Unruh, D., Müller-Quade, J.: Polynomial runtime and composability. Cryptology ePrint Archive, Report 2009/023 (2009). https://eprint.iacr.org/2009/023
23. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 456–486. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_15
24. Naor, M.: On cryptographic assumptions and challenges. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 96–109. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_6
25. Shoup, V.: Security analysis of SPAKE2+. Cryptology ePrint Archive, Report 2020/313 (2020). https://eprint.iacr.org/2020/313
26. Taubert, T., Wood, C.A.: SPAKE2+, an Augmented PAKE. Internet-Draft draft-bar-cfrg-spake2plus-00, Internet Engineering Task Force, March 2020. https://datatracker.ietf.org/doc/draft-bar-cfrg-spake2plus/